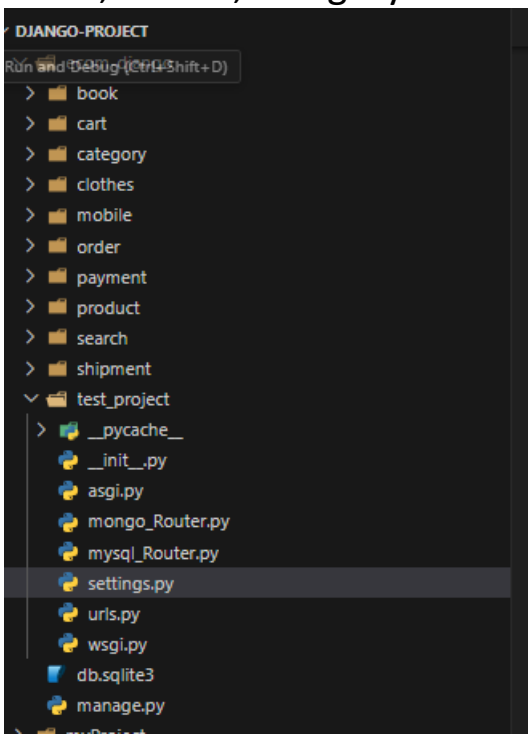
Bài tập 4:

1. Create app : book, mobile, clothes, payment, shipment, order, search, category



2. Khai báo app trong setings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'product',
    'cart',
    'category',
    'search',
    'shipment',
    'payment',
    'order',
    'mobile',
    'clothes',
    'book'
]
```

3. Config Database cho cả mongodb và mysqldb

```
DATABASES = {
    'default': {
        'ENGINE': 'djongo',
        'NAME': "ecom_mongodb",
    },
    'mysql_db': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'ecom_mysql',
        'USER': 'root',
        'PASSWORD': 'tuananhhalong123',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
DATABASE_ROUTERS = [
    'test_project.mysql_Router.MySqlRouter',
]
```

4. Tạo Router để điều hướng dữ liệu tới database

```python
class MySqlRouter:
    def db_for_read(self, model, **hints):
        if model._meta.app_label == 'cart'or model._meta.app_label == 'shipment' or model._meta.app_label == 'order':
            return 'mysql_db'
        elif model._meta.app_label == 'product' or model._meta.app_label == 'category' or model._meta.app_label == 'mobile':
            return 'default'
        # return 'default'

    def db_for_write(self, model, **hints):
        if model._meta.app_label == 'cart':
            return 'mysql_db'
        elif model._meta.app_label == 'product' or model._meta.app_label == 'category' or model._meta.app_label == 'mobile' :
            return 'default'
        # return 'default'

    def allow_relation(self, obj1, obj2, **hints):
        if obj1._meta.app_label == 'cart' or obj2._meta.app_label == 'cart':
            return True
        elif 'cart' not in [obj1._meta.app_label, obj2._meta.app_label]:
            return True
        elif obj1._meta.app_label == 'product' or obj2._meta.app_label == 'product':
            return True
        elif 'userchecking' not in [obj1._meta.app_label, obj2._meta.app_label]:
            return True
        elif obj1._meta.app_label == 'category' or obj2._meta.app_label == 'category':
            return True
        elif obj1._meta.app_label == 'mobile' or obj2._meta.app_label == 'mobile':
            return True
        elif 'category' not in [obj1._meta.app_label, obj2._meta.app_label]:
            return True
        return False

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        if app_label == 'cart' or app_label == 'shipment' or app_label == 'payment' or app_label == 'order':
            return db == 'mysql_db'
```

5 Code model.py , views.py , urls.py cho app Book

5.1 Model.py

```python
class Book(models.Model):
    category = models.ForeignKey(Category_Book, on_delete=models.CASCADE,null=True)
    title = models.CharField(max_length=255)
    author = models.CharField(max_length=255)
    description = models.TextField()
    price = models.DecimalField(default=0.0,max_digits=10, decimal_places=2)
    created_at = models.DateTimeField(default=timezone.now)
    updated_at = models.DateTimeField(auto_now=True)
```

5.2 views.py

```python
# Create your views here.
@api_view(['POST']) # neu khong co se bi error : 403 Forbidden
def create_book(request):
    serializer = BookSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET'])
def get_all(request):
        books = Book.objects.all()
        serializer = BookSerializer(books, many=True)
        return Response(serializer.data)
```

```python
@api_view(['GET'])
def get_byId(request,book_id):
        try:
            category = BookSerializer(Book.objects.get(pk=book_id)) # this is used when you want to convert a m
            # category = CategorySerializer(data=Category.objects.get(pk=category_id)) #his usage suggests that
            return Response(category.data)
        except Book.DoesNotExist:
            return Response('book does not exist', status=status.HTTP_400_BAD_REQUEST)

@api_view(['DELETE'])
def delete_book(request,book_id):
        book = Book.objects.get(pk=book_id)
        if not book:
            return Response(f'book with id={book_id} not exist',status=status.HTTP_404_NOT_FOUND)
        book.delete()
        return Response(f"Successfully deleted book with id={book_id}",status=status.HTTP_204_NO_CONTENT)
```

```python
@api_view(['PUT'])
def update_book(request, book_id):
    book = Book.objects.get(pk=book_id)
    if not book:
        print('not founddd')
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = BookSerializer(book, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

5.3 urls.py

```python
urlpatterns = [
    path('book/get_all', get_all, name='book_list'),
    path('book/detail/<str:book_id>', get_byId, name='book_detail'),
    path('book/create',create_book, name='create_book'),
    path('book/update/<str:book_id>', update_book, name='update_book'),
    path('book/delete/<str:book_id>', delete_book, name='delete_book'),
```

6.Code model.py , views.py , urls.py cho app Clothes

6.1model.py

```python
class Clothes(models.Model):
    category = models.ForeignKey(Category_Clothes, on_delete=models.CASCADE, null=True)
    title = models.CharField(max_length=255)
    brand = models.CharField(max_length=255)
    description = models.TextField()
    price = models.DecimalField(default=0.0, max_digits=10, decimal_places=2)
    created_at = models.DateTimeField(default=timezone.now)
    updated_at = models.DateTimeField(auto_now=True)
```

6.2 views.py

```python
@api_view(['POST']) # neu khong co se bi error : 403 Forbidden
def create_clothes(request):
    serializer = ClothesSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET'])
def get_all_clothes(request):
        clothes = Clothes.objects.all()
        serializer = ClothesSerializer(clothes, many=True)
        return Response(serializer.data)


@api_view(['GET'])
def get_byId_clothes(request,clothes_id):
        try:
            clothes = ClothesSerializer(Clothes.objects.get(pk=clothes_id)) # this is used wher
            # category = CategorySerializer(data=Category.objects.get(pk=category_id)) #his usa
            return Response(clothes.data)
        except Clothes.DoesNotExist:
            return Response('book does not exist', status=status.HTTP_400_BAD_REQUEST)
```

```python
@api_view(['DELETE'])
def delete_clothes(request,clothes_id):
        clothes = Clothes.objects.get(pk=clothes_id)
        if not clothes:
            return Response(f'book with id={clothes_id} not exist',status=status.HTTP_404_NOT_FOUND)
        clothes.delete()
        return Response(f"Successfully deleted book with id={clothes_id}",status=status.HTTP_204_NO_CONTENT)

@api_view(['PUT'])
def update_clothes(request, clothes_id):
    clothes = Clothes.objects.get(pk=clothes_id)
    if not clothes:
        print('not founddd')
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = ClothesSerializer(clothes, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

6.3 urls.py

```python
  path('clothes/get_all', get_all_clothes, name='clothes_list'),
  path('clothes/detail/<str:clothes_id>', get_byId_clothes, name='clothes_detail'),
  path('clothes/create',create_clothes, name='create_clothes'),
  path('clothes/update/<str:clothes_id>', update_clothes, name='update_clothes'),
  path('clothes/delete/<str:clothes_id>', delete_clothes, name='delete_clothes'),
```

7. Code model.py , views.py , urls.py cho app Clothes

7.1 model.py

```python
class Mobile(models.Model):
    brand = models.CharField(max_length=255)
    price = models.DecimalField(default=0.0, max_digits=10, decimal_places=2)
    description = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

7.2 views.py

```python
@api_view(['POST']) # neu khong co se bi error : 403 Forbidden
def create_clothes(request):
    serializer = ClothesSerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET'])
def get_all_clothes(request):
        clothes = Clothes.objects.all()
        serializer = ClothesSerializer(clothes, many=True)
        return Response(serializer.data)


@api_view(['GET'])
def get_byId_clothes(request,clothes_id):
        try:
            clothes = ClothesSerializer(Clothes.objects.get(pk=clothes_id)) # this is used when
            # category = CategorySerializer(data=Category.objects.get(pk=category_id)) #his usa
            return Response(clothes.data)
        except Clothes.DoesNotExist:
            return Response('book does not exist', status=status.HTTP_400_BAD_REQUEST)
```

```python
@api_view(['DELETE'])
def delete_clothes(request,clothes_id):
        clothes = Clothes.objects.get(pk=clothes_id)
        if not clothes:
            return Response(f'book with id={clothes_id} not exist',status=status.HTTP_404_NOT_FOUND)
        clothes.delete()
        return Response(f"Successfully deleted book with id={clothes_id}",status=status.HTTP_204_NO_CONTENT)

@api_view(['PUT'])
def update_clothes(request, clothes_id):
    clothes = Clothes.objects.get(pk=clothes_id)
    if not clothes:
        print('not founddd')
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = ClothesSerializer(clothes, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

7.3 urls.py

```python
urlpatterns = [
    path('Mobile/get_all', get_all, name='Mobile_list'),
    path('Mobile/detail/<str:Mobile_id>', get_byId, name='Mobile_detail'),
    path('Mobile/create',create_Mobile, name='create_Mobile'),
    path('Mobile/update/<str:Mobile_id>', update_Mobile, name='update_Mobile'),
    path('Mobile/delete/<str:Mobile_id>', delete_Mobile, name='delete_Mobile'),
```

8.Code model.py , views.py , urls.py cho app Category

8.1 model.py

```python
# Create your models here.
class Category_Book(models.Model):
    name = models.CharField(max_length=255)
    description = models.CharField(max_length=255)

    def __str__(self):
            return self.name

class Category_Clothes(models.Model):
    name = models.CharField(max_length=255)
    description = models.CharField(max_length=255)

    def __str__(self):
            return self.name

class Category_Mobile(models.Model):
    name = models.CharField(max_length=255)
    description = models.CharField(max_length=255)

    def __str__(self):
            return self.name
```

8.2 views.py

```python
@api_view(['POST']) # neu khong co se bi error : 403 Forbidden
def create_category_book(request):
    serializer = CategorySerializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET'])
def get_all_book(request):
        books = Category_Book.objects.all()
        serializer = CategorySerializer(books, many=True)
        return Response(serializer.data)
```

```python
@api_view(['GET'])
def get_byId_book(request,category_id):
        try:
            category = CategorySerializer(Category_Book.objects.get(pk=category_id)) # this is used when you want to
            # category = CategorySerializer(data=Category.objects.get(pk=category_id)) #his usage suggests that the
            return Response(category.data)
        except Category_Book.DoesNotExist:
            return Response('Category does not exist', status=status.HTTP_400_BAD_REQUEST)

@api_view(['DELETE'])
def delete_category_book(request,category_id):
        category = Category_Book.objects.get(pk=category_id)
        if not category:
            return Response(f'category with id={category_id} not exist',status=status.HTTP_404_NOT_FOUND)
        category.delete()
        return Response(f"Successfully deleted category with id={category_id}",status=status.HTTP_204_NO_CONTENT)
```

```python
@api_view(['PUT'])
def update_category_book(request, category_id):
    category = Category_Book.objects.get(pk=category_id)
    if not category:
        print('not founddd')
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = CategorySerializer(category, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

```python
# Create your views here.
@api_view(['POST']) # neu khong co se bi error : 403 Forbidden
def create_category_clothes(request):
    serializer = Category_Clothes_Serializer(data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

@api_view(['GET'])
def get_all_category_clothes(request):
        books = Category_Clothes.objects.all()
        serializer = Category_Clothes_Serializer(books, many=True)
        return Response(serializer.data)
```

```python
@api_view(['GET'])
def get_byId_category_clothes(request,category_id):
        try:
            category = Category_Clothes(Category_Clothes.objects.get(pk=category_id)) # this is used when you want t
            # category = CategorySerializer(data=Category.objects.get(pk=category_id)) #his usage suggests that the
            return Response(category.data)
        except Category_Clothes.DoesNotExist:
            return Response('Category does not exist', status=status.HTTP_400_BAD_REQUEST)

@api_view(['DELETE'])
def delete_category_clothes(request,category_id):
        category = Category_Clothes.objects.get(pk=category_id)
        if not category:
            return Response(f'category with id={category_id} not exist',status=status.HTTP_404_NOT_FOUND)
        category.delete()
        return Response(f"Successfully deleted category with id={category_id}",status=status.HTTP_204_NO_CONTENT)
```

```python
@api_view(['PUT'])
def update_category_clothes(request, category_id):
    category = Category_Clothes.objects.get(pk=category_id)
    if not category:
        print('not founddd')
        return Response(status=status.HTTP_404_NOT_FOUND)
    serializer = Category_Clothes_Serializer(category, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

8.3 urls.py

```python
urlpatterns = [
    path('category_book/get_all', get_all_book, name='category_list_book'),
    path('category_book/detail/<str:category_id>', get_byId_book, name='category_detail_book'),
    path('category_book/create',create_category_book, name='create_category_book'),
    path('category_book/update/<str:category_id>', update_category_book, name='update_category_book'),
    path('category_book/delete/<str:category_id>', delete_category_book, name='delete_category_book'),


    path('category_clothes/get_all', get_all_category_clothes, name='category_list_clothes'),
    path('category_clothes/detail/<str:category_id>', get_byId_category_clothes, name='category_detail_clothes'),
    path('category_clothes/create',create_category_clothes, name='create_category'),
    path('category_clothes/update/<str:category_id>', update_category_clothes, name='update_category_clothes'),
    path('category_clothes/delete/<str:category_id>', delete_category_clothes, name='delete_category_clothes'),

]
```

9.views.py trong app search

```python
@api_view(['GET'])
def search_book(request):
        # query = request.GET.get('title') #lấy dữ liệu từ string query
        query = request.data.get('title') # lấy dữ liệu từ payload
        print(query)
        if query:
            books = Book.objects.filter(title__icontains=query)
            serializer = BookSerializer(books, many=True)
            return Response(serializer.data)
        else:
            return Response('Book does not exist', status=status.HTTP_400_BAD_REQUEST)
```

```python
urlpatterns = [
    path('book/search', search_book, name='search_book'),
]
```

10. Code model.py , views.py , urls.py cho app Cart

10.1 model.py

```python
# Create your models here.
class Cart(models.Model):
    user_id = models.CharField(max_length=24)
    quantity = models.PositiveIntegerField(default=1)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.user.username}'s Cart"

class CartItem(models.Model):
    product_id = models.CharField(max_length=24)
    cart = models.ForeignKey(Cart, on_delete=models.CASCADE)
    #Cart: This is the model class that the foreign key is referring to. In this case, Cart is the target model, and
    #on_delete=models.CASCADE: This parameter specifies the behavior to follow when the referenced Cart instance is
```

10.2 views.py

```python
# Create your views here.
@api_view(['POST'])
def add_book_to_cart(request):
    try:
        user_id = request.data.get('user_id')
        book_id = request.data.get('book_id')

        # Kiểm tra xem trong bảng Cart đã tồn tại cart của người dùng hay chưa
        try:
            cart = Cart.objects.get(user_id=user_id)
            cart.quantity += 1
            cart.save()
            cart_item = CartItem.objects.create(product_id=book_id, cart=cart)
            created = False
        except Cart.DoesNotExist:
            # Tạo mới cart nếu chưa tồn tại
            print(222222222)
            cart = Cart.objects.create(user_id=user_id, quantity=1)
            cart_item = CartItem.objects.create(product_id=book_id, cart=cart)
            created = True

        print(user_id+"  "+book_id)
        if created:
            message = 'Cart created and book added successfully'
        else:
            message = 'Book added to the cart successfully'

        return Response({'message': message})
    except Exception as e:
        return Response({'error': str(e)}, status=400)
```

10.3 urls.py

```python
urlpatterns = [
    path('cart/add_book_to_cart', add_book_to_cart, name='add_book_to_cart'),
]
```

11. Code model.py , views.py , urls.py cho app shipment

11.1 model.py

```
# Create your models here.
class shipment(models.Model):
    ### The following are the fields of our table.
    fname = models.CharField(max_length=50)
    lname = models.CharField(max_length=50)
    email = models.CharField(max_length=50)
    mobile = models.CharField(max_length=12)
    address = models.CharField(max_length=200)
    product_id = models.CharField(max_length=10)
    quantity = models.CharField(max_length=5)
    payment_status = models.CharField(max_length=15)
    transaction_id = models.CharField(max_length=5)
    shipment_status = models.CharField(max_length=20)
    ### It will help to print the values.
    def __str__(self):
        return '%s %s %s %s %s %s %s %s %s %s' % (self.fname, self.
        lname, self.email, self.mobile, self.product_id, self.address, self.
        quantity , self.payment_status, self.transaction_id, self.shipment_status)
```

11.2 views.py

```
### This function is inserting the data into our table.
def ship_data_insert(fname, lname, email, mobile, address, product_id,
quantity, payment_status, transaction_id, shipment_status):
    shipment_data = ship_obj(fname = fname,lname = lname, email = email,
    mobile = mobile,
    address = address, product_id = product_id, quantity = quantity,
    payment_status = payment_status, transaction_id = transaction_id,
    shipment_status = shipment_status)
    shipment_data.save()
    return 1
```

```python
### This function will get the data from the front end.
@csrf_exempt
def shipment_reg_update(request):
    if request.method == 'POST':
        if 'application/json' in request.META['CONTENT_TYPE']:
            val1 = json.loads(request.body)
            ### This is for reading the inputs from JSON.
            fname = val1.get("First Name")
            lname = val1.get("Last Name")
            email = val1.get("Email Id")
            mobile = val1.get("Mobile Number")
            address = val1.get("Address")
            product_id = val1.get("Product Id")
            quantity = val1.get("Quantity")
            payment_status = val1.get("Payment Status")
            transaction_id = val1.get("Transaction Id")
            shipment_status = "ready to dispatch"

            resp = {}
            ### After all validation, it will call the data_insert function.
            respdata = ship_data_insert(fname, lname, email, mobile,
            address, product_id, quantity, payment_status, transaction_id, shipment_status)
        ### If it returns value then will show success.
            if respdata:
                resp['status'] = 'Success'
                resp['status_code'] = '200'
                resp['message'] = 'Product is ready to dispatch.'
            ### If value is not found then it will give failed in response.
            else:
                resp['status'] = 'Failed'
                resp['status_code'] = '400'
                resp['message'] = 'Failed to update shipment details.'
    return HttpResponse(json.dumps(resp), content_type = 'application/json')
```

```python
### This function is used for finding the transaction.
def shipment_data(uname):
    data = ship_obj.objects.filter(email = uname)
    for val in data.values():
        return val


### This function is used for getting the shipment status
@csrf_exempt
def shipment_status(request):
    if request.method == 'POST':
        if 'application/json' in request.META['CONTENT_TYPE']:
            variable1 = json.loads(request.body)
            ### This is for reading the inputs from JSON.
            uname = variable1.get("User Name")
            resp = {}
            ### It will call the shipment_data function.
            respdata = shipment_data(uname)
            ### If it returns value then will show success.
            if respdata:
                resp['status'] = 'Success'
                resp['status_code'] = '200'
                resp['message'] = respdata
                ### If it is not returning any value then it will show failed response.
            else:
                resp['status'] = 'Failed'
                resp['status_code'] = '400'
                resp['message'] = 'User data is not available.'
    return HttpResponse(json.dumps(resp), content_type = 'application/json')
```

11.3 urls.py

```python
urlpatterns = [
    path('api/shipment/shipment_status', shipment_status, name='shipment_status'),
    path('api/shipment/shipment_updates', shipment_reg_update, name='shipment_reg_update'),

]
```

12. Code model.py , views.py , urls.py cho app Payment

12.1 Model

```
# This is our model for user registration.
class payment_status(models.Model):
    ### The following are the fields of our table.
    username = models.CharField(max_length=10)
    product_id = models.CharField(max_length=10)
    price = models.CharField(max_length=10)
    quantity = models.CharField(max_length=5)
    mode_of_payment = models.CharField(max_length=20)
    mobile = models.CharField(max_length=12)
    status = models.CharField(max_length=15)
    ### It will help to print the values.
    def __str__(self):
        return '%s %s %s %s %s %s %s ' % (self.username, self.product_id, self.price, self.quantity,
```

12.2 views.py

```
### This function is for fetching the user data.
def get_transaction_details(uname):
    user = paystat.objects.filter(username = uname)
    for data in user.values():
        return data

### This function is used for storing the data.
def store_data(uname, prodid, price, quantity, mode_of_payment, mobile):
    user_data = paystat(username = uname, product_id = prodid, price =
    price, quantity = quantity, mode_of_payment = mode_of_payment, mobile =
    mobile, status = "Success")

    user_data.save()
    return 1
```

```python
### This function is created for getting the payment.
@csrf_exempt
def get_payment(request):
    uname = request.POST.get("User name")
    prodid = request.POST.get("Product id")
    price = request.POST.get("Product price")
    quantity = request.POST.get("Product quantity")
    mode_of_payment = request.POST.get("Payment mode")
    mobile = request.POST.get("Mobile Number")
    print(uname)
    print(prodid)
    print(price)
    print(quantity)
    print(mode_of_payment)
    print(mobile)
    resp = {}
    if uname and prodid and price and quantity and mode_of_payment and  mobile:
        ### It will call the store data function.
        respdata = store_data(uname, prodid, price, quantity, mode_of_payment, mobile)
        respdata2 = ship_update(uname)
        ### If it returns value then will show success.
        if respdata:
            resp['status'] = 'Success'
            resp['status_code'] = '200'
            resp['message'] = 'Transaction is completed.'
        ### If it is returning null value then it will show failed.
        else:
            resp['status'] = 'Failed'
            resp['status_code'] = '400'
            resp['message'] = 'Transaction is failed, Please try again.'
        ### If any mandatory field is missing then it will be through a failed message.
    else:
        resp['status'] = 'Failed'
        resp['status_code'] = '400'
        resp['message'] = 'All fields are mandatory.'
    return HttpResponse(json.dumps(resp), content_type = 'application/json')
```

```python
### This function is created for getting the username and password.
@csrf_exempt
def user_transaction_info(request):
    # uname = request.POST.get("User Name")
    if request.method == 'POST':
        if 'application/json' in request.META['CONTENT_TYPE']:
            val1 = json.loads(request.body)
            uname = val1.get('User Name')
            resp = {}
            if uname:
                ## Calling the getting the user info.
                respdata = get_transaction_details(uname)
                if respdata:
                    resp['status'] = 'Success'
                    resp['status_code'] = '200'
                    resp['data'] = respdata
                 ### If a user is not found then it give failed as response.
                else:
                    resp['status'] = 'Failed'
                    resp['status_code'] = '400'
                    resp['message'] = 'User Not Found.'
            ### The field value is missing.
            else:
                resp['status'] = 'Failed'
                resp['status_code'] = '400'
                resp['message'] = 'Fields is mandatory.'
        else:
            resp['status'] = 'Failed'
            resp['status_code'] = '400'
            resp['message'] = 'Request type is not matched.'
    else:
        resp['status'] = 'Failed'
        resp['status_code'] = '400'
        resp['message'] = 'Request type is not matched.'
    return HttpResponse(json.dumps(resp), content_type = 'application/son')
```

12.3 urls.py

```
urlpatterns = [
    path('api/payment/initiate_payment', get_payment, name='get_payment'),
    path('api/payment/get_transaction_info', user_transaction_info, name='get_transaction_info'),
]
```

CÂU 1--5 MÔN SOFTWARE ARCHITECTURE AND DESIGN

Câu 1:

1.1 .Trình bày bằng ngôn ngữ tự nhiên

Hệ thống quản lý thương mại điện tử EcoMaS (E-commerce Management System) là một giải pháp phần mềm toàn diện được phát triển để hỗ trợ các doanh nghiệp và cá nhân trong việc xây dựng, quản lý và vận hành cửa hàng trực tuyến của họ. Được xây dựng trên nền tảng công nghệ hiện đại, EcoMaS cung cấp một loạt các tính năng quản lý và khả năng tương tác giúp tối ưu hóa hoạt động kinh doanh thương mại điện tử.

• Quản lý Sản phẩm và Dịch vụ: EcoMaS cho phép người dùng quản lý một danh mục đa dạng của sản phẩm và dịch vụ. Người dùng có thể thêm, chỉnh sửa hoặc xóa các thông tin liên quan đến sản phẩm như tên, mô tả, hình ảnh, giá cả, mã sản phẩm và thông tin khác. Hệ thống cung cấp giao diện dễ sử dụng để tải lên hình ảnh sản phẩm và cung cấp thông tin chi tiết để giới thiệu sản phẩm một cách hấp dẫn.

• Quản lý Đơn đặt hàng: EcoMaS cho phép người dùng theo dõi và quản lý các đơn đặt hàng từ khách hàng. Người dùng có thể xem thông tin chi tiết về đơn hàng, bao gồm sản phẩm, số lượng, giá cả và thông tin giao hàng. Hệ thống cung cấp khả năng cập nhật trạng thái của đơn hàng và thông báo cho khách hàng về tình trạng vận chuyển và giao hàng.

• Quản lý Tồn kho: EcoMaS hỗ trợ việc quản lý tồn kho của sản phẩm. Người dùng có thể theo dõi số lượng tồn kho hiện tại của mỗi sản phẩm và nhận thông báo khi tồn kho cạn kiệt. Điều này giúp người dùng duy trì sự cân đối giữa cung cấp và cầu cung của sản phẩm.

• Thanh toán trực tuyến: Hệ thống cho phép khách hàng thực hiện thanh toán trực tuyến an toàn và thuận tiện thông qua các phương thức thanh toán khác nhau như thẻ tín dụng, chuyển khoản ngân hàng hoặc ví điện tử. Thông tin thanh toán được bảo vệ bằng các biện pháp bảo mật chống lại việc lạm dụng.

• Tương tác với Khách hàng: EcoMaS cung cấp khả năng tạo tương tác giữa người dùng và khách hàng thông qua hệ thống tin nhắn nội bộ. Khách hàng có thể đặt câu hỏi, yêu cầu hỗ trợ hoặc gửi phản hồi và người dùng có thể phản hồi một cách hiệu quả.

• Phân quyền và Bảo mật: Hệ thống hỗ trợ quản lý phân quyền, giúp người dùng kiểm soát quyền truy cập và chức năng tương ứng với từng vai trò khác nhau trong tổ chức. Điều này đảm bảo an toàn thông tin và hoạt động kinh doanh

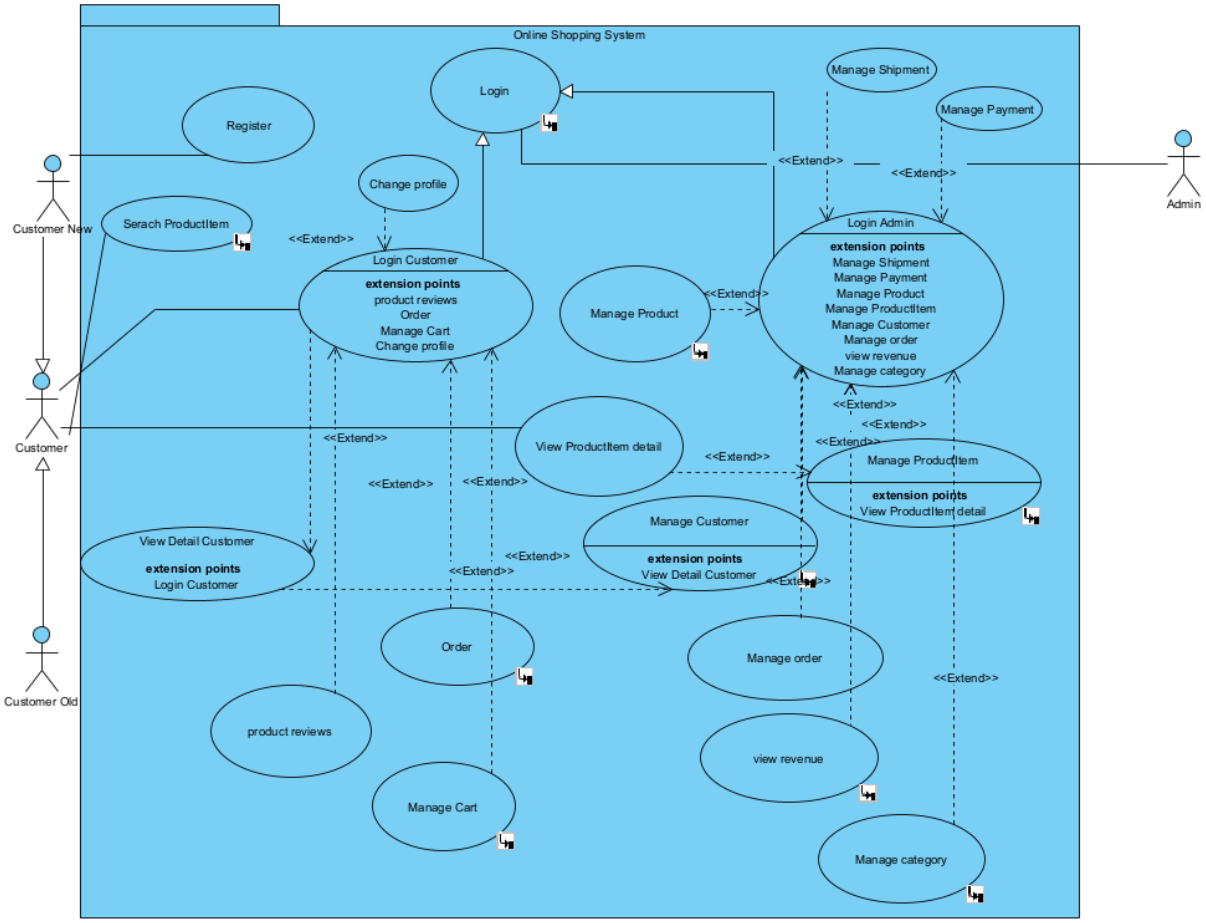
2. Mô tả các actor (primary & secondary actor) của hệ thống
2.1. Primary actor1. Khách hàng (Customer): Đây là người dùng cuối, họ là những người mua sắm sản phẩm hoặc dịch vụ từ các cửa hàng trực tuyến. Khách hàng truy cập vào hệ thống để tìm kiếm, xem sản phẩm, thêm vào giỏ hàng, thực hiện thanh toán và quản lý đơn hàng của họ.
2. Chủ cửa hàng (Store Owner): Chủ cửa hàng là người quản lý cửa hàng trực tuyến. Họ có quyền quản lý danh mục sản phẩm, giá cả, mô tả, hình ảnh sản phẩm và thực hiện các tác vụ quản lý tồn kho. Chủ cửa hàng cũng có khả năng xem và quản lý các đơn đặt hàng từ khách hàng.
2.2. Secondary Actors
1. Nhà cung cấp (Supplier): Nhà cung cấp là các đối tác cung cấp sản phẩm hoặc dịch vụ cho cửa hàng thương mại điện tử. Họ có thể cung cấp thông tin về sản phẩm, số lượng tồn kho, giá cả và các chi tiết khác cho cửa hàng.
2. Dịch vụ giao hàng (Delivery Service): Đây là các dịch vụ vận chuyển và giao hàng chịu trách nhiệm vận chuyển các sản phẩm từ cửa hàng đến khách hàng cuối. Họ có thể liên quan đến việc cập nhật thông tin vận chuyển và trạng thái giao hàng.
3. Hệ thống thanh toán (Payment System): Hệ thống thanh toán là những dịch vụ xử lý thanh toán trực tuyến, đảm bảo rằng các giao dịch thanh toán được thực hiện một cách an toàn và hiệu quả.
4. Hỗ trợ khách hàng (Customer Support): Người tham gia này cung cấp hỗ trợ cho khách hàng về các vấn đề liên quan đến sản phẩm, đơn hàng, thanh toán hoặc bất kỳ vấn đề nào khác liên quan đến việc sử dụng hệ thống.
5. Hệ thống quản lý (Management System): Đây là các hệ thống hỗ trợ quản lý tổng thể của cửa hàng, bao gồm quản lý dữ liệu sản phẩm, tồn kho, đơn hàng và thông tin khách hàng.
3. Mô tả yêu cầu các chức năng và phi chức năng
3.1. Các yêu cầu chức năng
1. Quản lý Sản phẩm: Hệ thống cần cho phép chủ cửa hàng thêm, sửa đổi và xóa các thông tin liên quan đến sản phẩm, bao gồm tên, mô tả, hình ảnh, giá cả và số lượng tồn kho.
 Thêm sản phẩm: Chủ cửa hàng thêm một sản phẩm mới bằng cách cung cấp các thông tin như tên, mô tả, giá cả, số lượng tồn kho và hình ảnh. Hệ thống lưu thông tin và cập nhật danh sách sản phẩm.
 Sửa thông tin: Chủ cửa hàng có thể sửa đổi thông tin của sản phẩm như tên, mô tả, giá cả, số lượng tồn kho và hình ảnh. Thay đổi này được cập nhật trong cơ sở dữ liệu sản phẩm.
 Xóa sản phẩm: Chủ cửa hàng có thể xóa sản phẩm khỏi danh sách. Hệ thống sẽ xác nhận việc xóa trước khi thực hiện. Việc xóa sản phẩm cần cân nhắc kỹ lưỡng vì nó có thể ảnh hưởng đến lịch sử đơn hàng và dữ liệu khách hàng. - Hiển thị danh sách: Hệ thống hiển thị danh sách tất cả các sản phẩm bao gồm tên, giá cả và số lượng tồn kho. Người dùng có thể sắp xếp danh sách theo tên hoặc giá cả để dễ dàng tìm kiếm

 Tìm kiếm sản phẩm: Chức năng tìm kiếm cho phép người dùng tìm kiếm sản phẩm bằng cách nhập từ khóa hoặc tiêu chí tìm kiếm. Hệ thống sẽ hiển thị các sản phẩm phù hợp với yêu cầu tìm kiếm.
2. Quản lý Đơn đặt hàng: Hệ thống phải giúp người dùng quản lý các đơn đặt hàng từ khách hàng. Người dùng cần có khả năng xem thông tin chi tiết của đơn hàng, cập nhật trạng thái đơn hàng và thông báo cho khách hàng về tình trạng vận chuyển.
 Xem chi tiết đơn đặt hàng: Người dùng có thể xem thông tin chi tiết của đơn đặt hàng bao gồm danh sách sản phẩm, giá cả, số lượng và tổng cộng. Họ cũng có thể xem trạng thái vận chuyển và thông tin người nhận.
 Cập nhật trạng thái đơn hàng: Chủ cửa hàng có khả năng cập nhật trạng thái của đơn đặt hàng, chẳng hạn như "Đã xác nhận", "Đang vận chuyển", "Đã giao hàng". Thay đổi trạng thái này cần được cập nhật liên tục để khách hàng biết được tình trạng đơn hàng của mình.
 Thông báo cho khách hàng: Hệ thống có khả năng gửi thông báo về trạng thái vận chuyển và giao hàng cho khách hàng qua email hoặc tin nhắn. Điều này giúp khách hàng theo dõi tình trạng đơn hàng và chuẩn bị sẵn sàng cho việc nhận hàng.
3. Thanh toán và Đơn hàng: Hệ thống cần hỗ trợ thanh toán trực tuyến cho khách hàng thông qua nhiều phương thức thanh toán khác nhau và phải đảm bảo tính bảo mật của thông tin thanh toán. Sau khi thanh toán, hệ thống cần cập nhật trạng thái của đơn hàng và thông báo cho khách hàng về tình trạng vận chuyển.
 Thanh toán trực tuyến: Khách hàng chọn sản phẩm, thêm vào giỏ hàng và tiến hành thanh toán trực tuyến. Hệ thống xử lý giao dịch thanh toán qua các phương thức như thẻ tín dụng, ví điện tử hoặc chuyển khoản.
 Cập nhật trạng thái đơn hàng: Sau khi khách hàng hoàn thành thanh toán, hệ thống tự động cập nhật trạng thái đơn hàng của họ. Điều này đảm bảo rằng cả khách hàng và chủ cửa hàng đều biết được tình trạng của đơn hàng.
4. Quản lý Tồn kho: Hệ thống cần cho phép quản lý tồn kho của sản phẩm. Người dùng cần thể hiện số lượng tồn kho hiện tại của từng sản phẩm và nhận thông báo khi tồn kho gần hết.
 Xem tồn kho: Chức năng này cho phép chủ cửa hàng xem tổng số lượng tồn kho của mỗi sản phẩm. Thông tin tồn kho giúp quản lý cung cấp và đảm bảo sẵn sàng sản phẩm cho đơn hàng.
 -Thông báo tồn kho: Hệ thống có khả năng gửi thông báo đến chủ cửa hàng khi tồn kho của sản phẩm nào đó gần hết. Điều này giúp chủ cửa hàng có thời gian chuẩn bị thêm hàng mới.
3.2. Các yêu cầu phi chức năng của hệ thống :
1. Hiệu suất và Tải trọng: Hệ thống phải xử lý nhiều người dùng cùng lúc mà không gây chậm trễ Xử lý đồng thời lượng lớn người dùng: Hệ thống cần xử lý một số lượng lớn người dùng đồng thời mà không gây trễ hoặc sự cố. Điều này đảm bảo trải nghiệm mua sắm mượt mà cho tất cả người dùng.
2. Bảo mật và Quyền truy cập: Dữ liệu khách hàng, thanh toán và quản lý cần được bảo mật. Chỉ người
dùng có quyền mới có thể truy cập và quản lý thông tin. -Bảo mật Dữ liệu: Hệ thống cần đảm bảo tính bảo mật cho dữ liệu khách hàng, thông tin thanh toán và thông tin quản lý. Điều này đảm bảo không có người không được phép truy cập vào thông tin nhạy cảm.

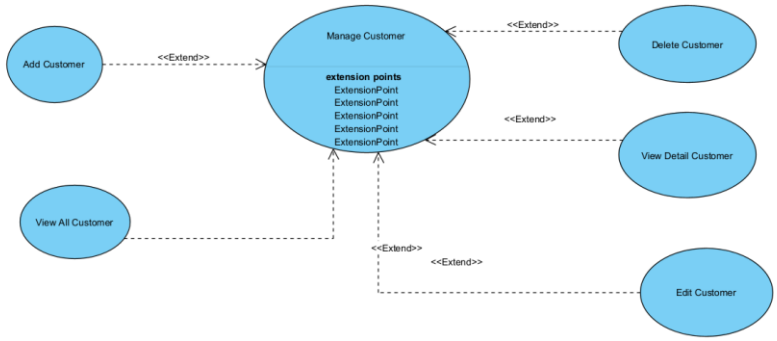
-Quản lý Quyền truy cập: Hệ thống cần thực hiện quản lý quyền truy cập sao cho chỉ những người dùng được phép mới có thể truy cập và thực hiện các chức năng quản lý như thêm, sửa, xóa sản phẩm

Câu 2:

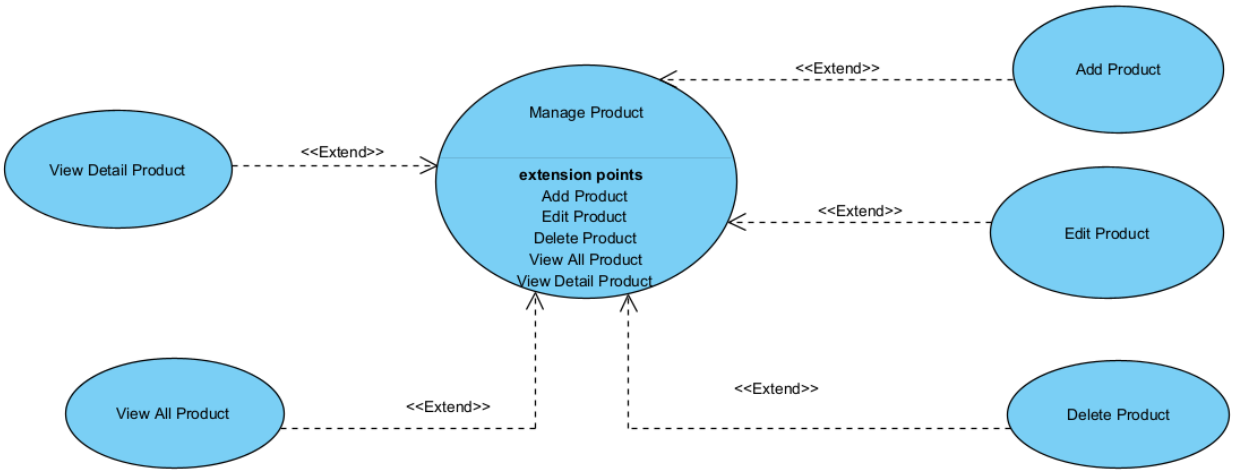
1 Overal Usecase Diagram



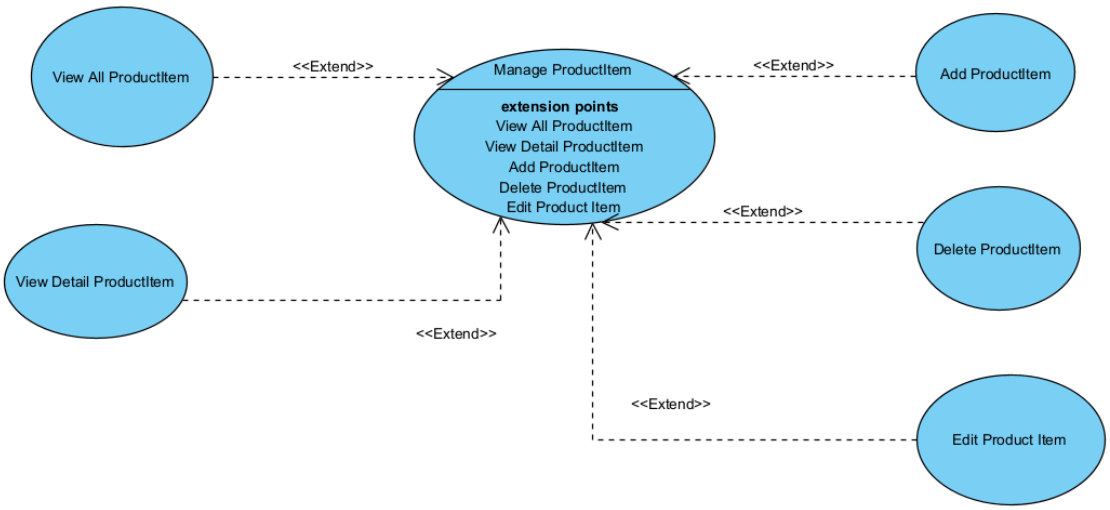
2 Manage Customer Use Case Diagram



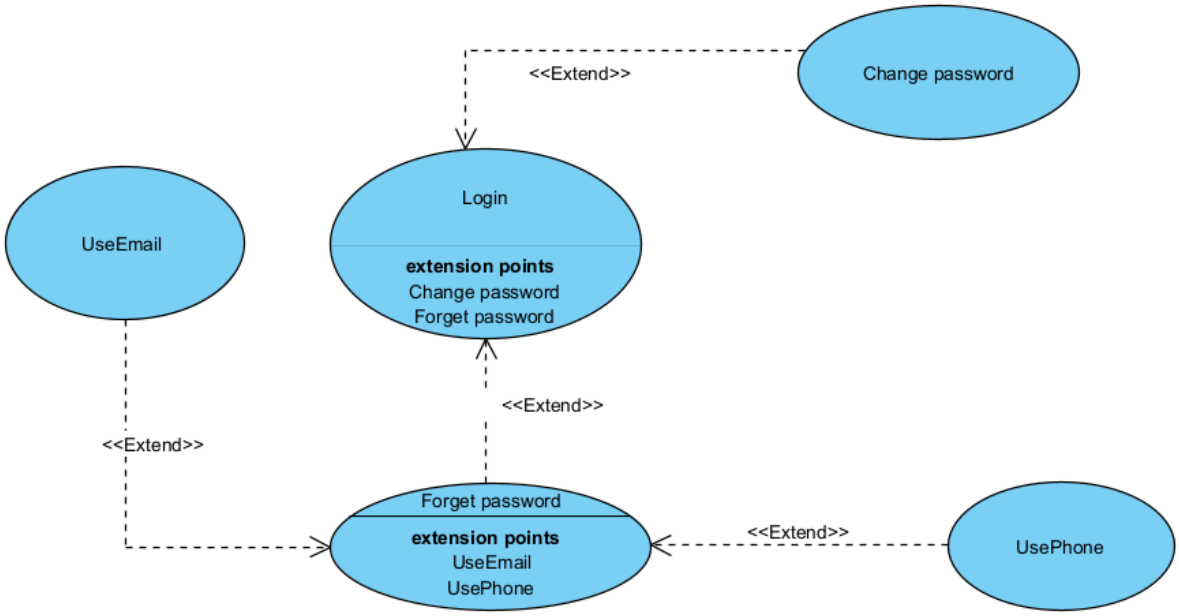
3 Manage Product Use Case Diagram



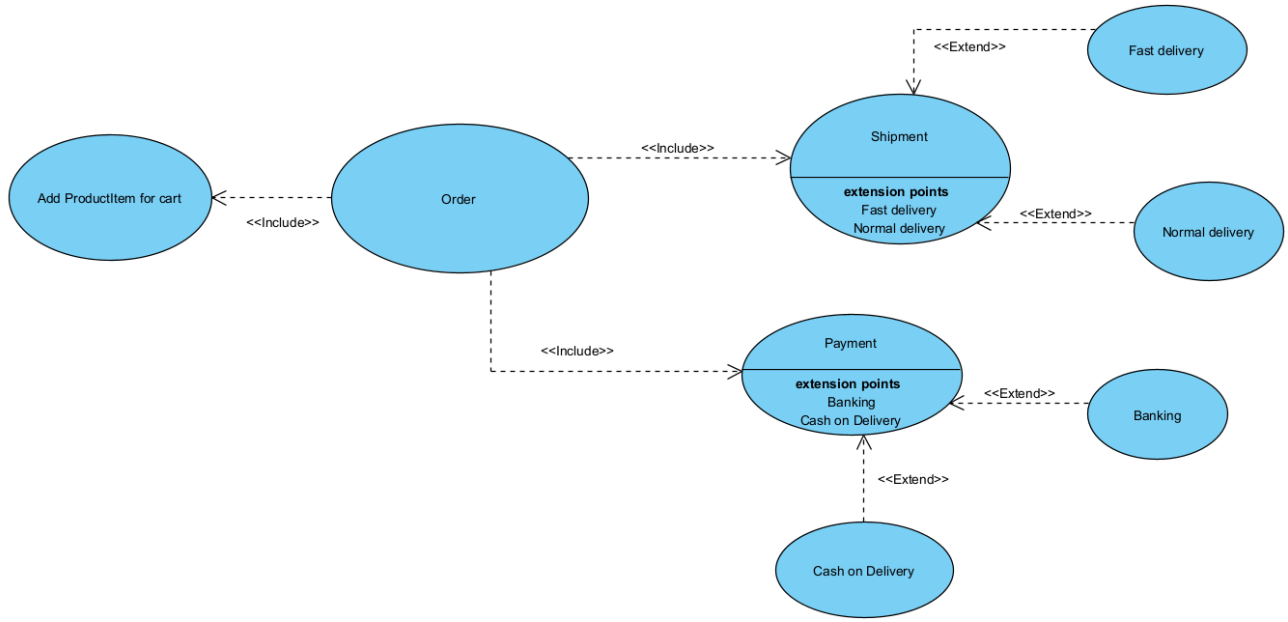
4 Manage ProductItem Use Case Diagram



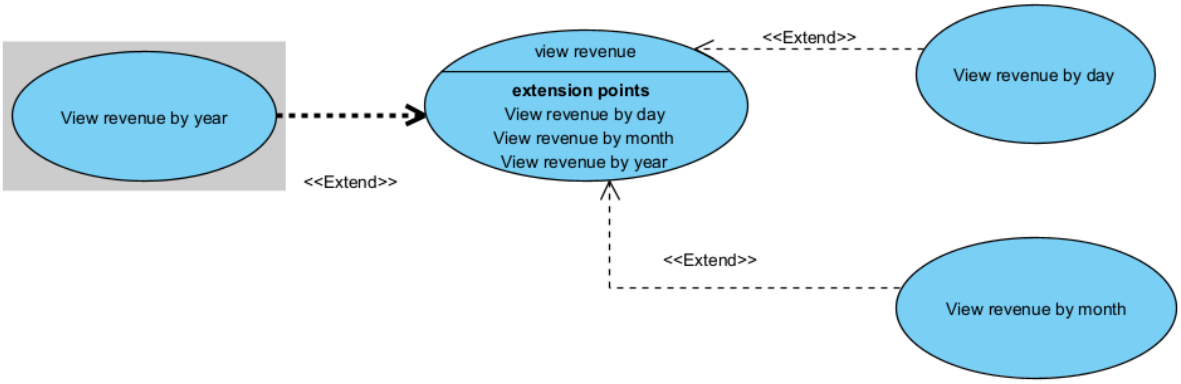
5 Login Use Case Diagram



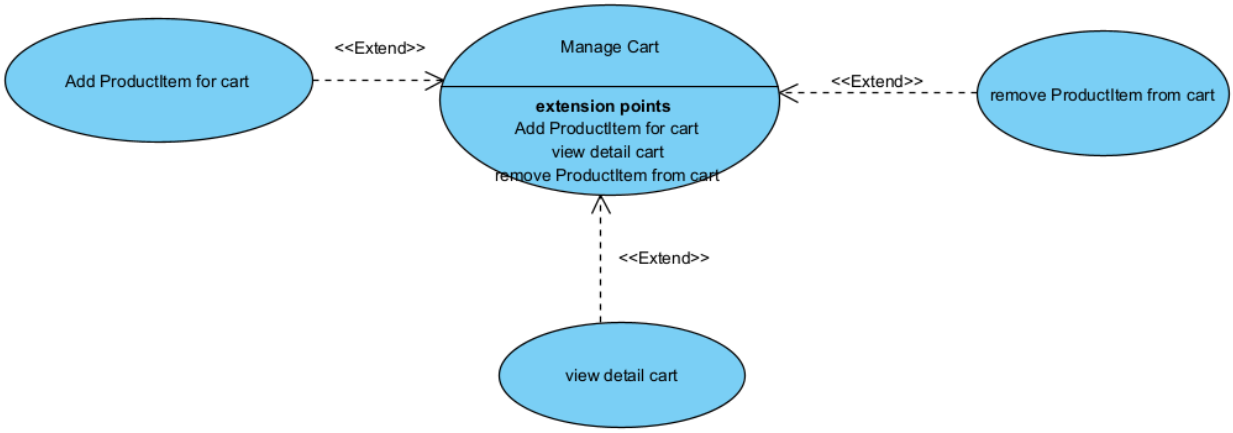
6 Order Use Case Diagram



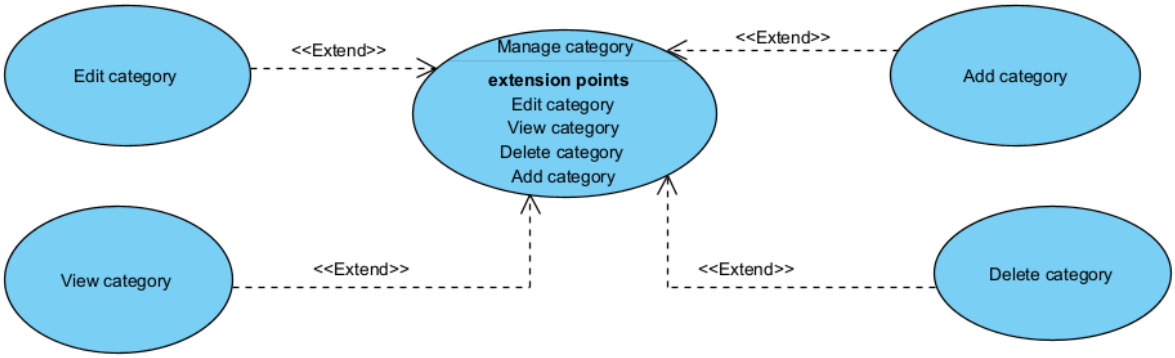
7 view revenue Use Case Diagram



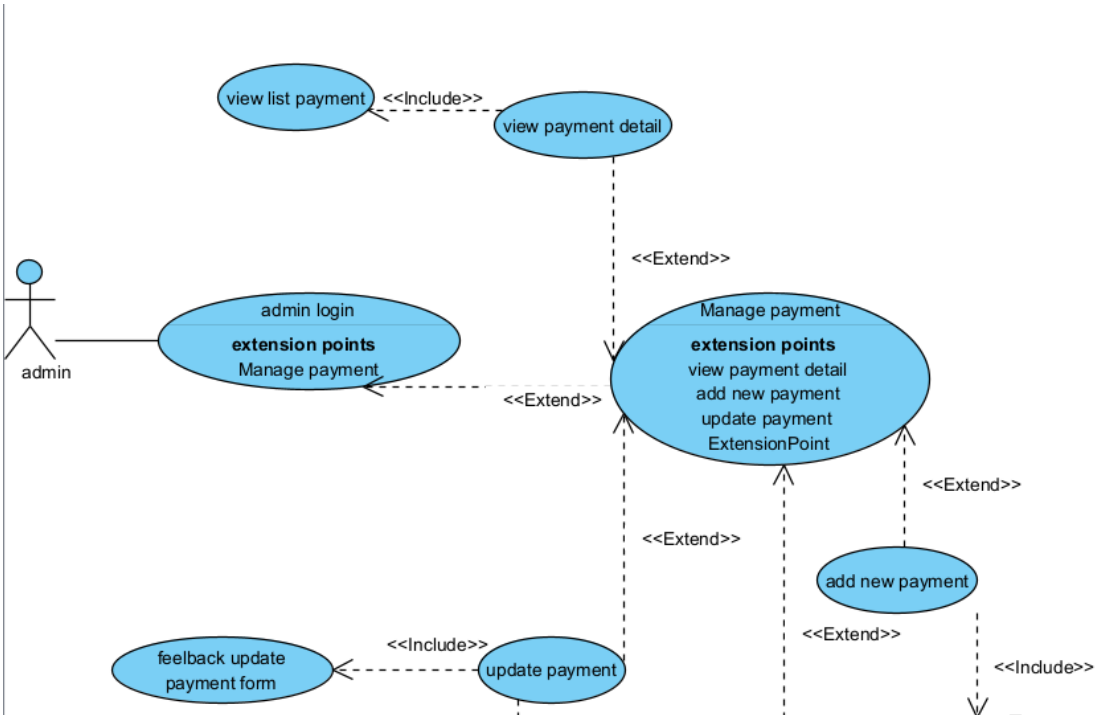
8 Manage Cart Use Case Diagram



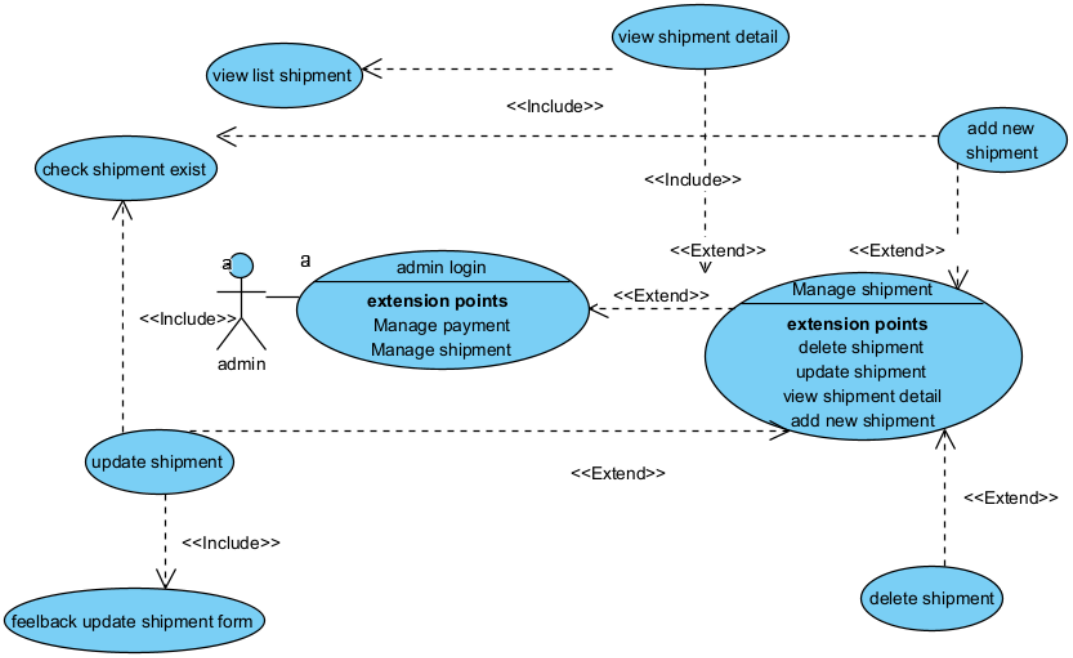
9 Manage Category Use Case Diagram



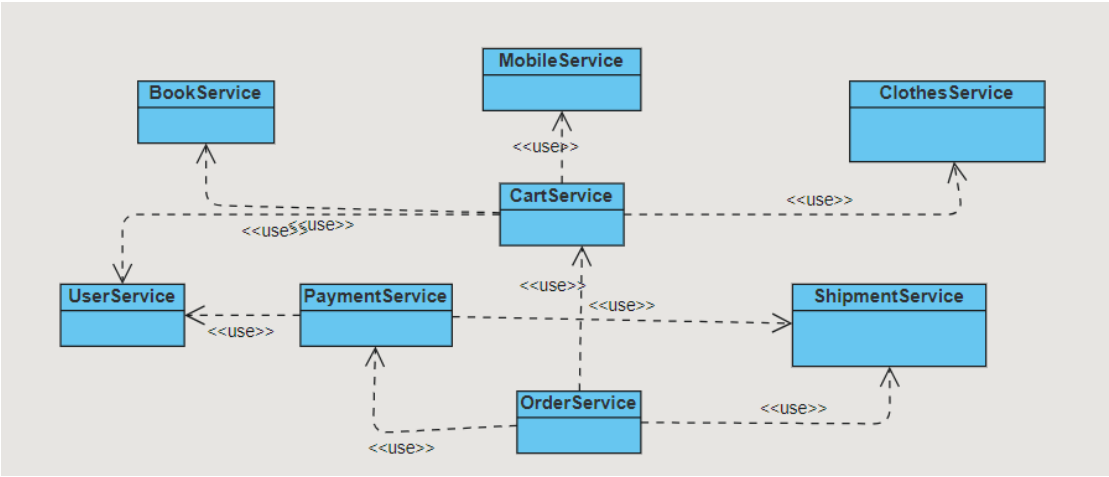
10 Manage Payment Use Case Diagram



11 Manage Shipment Use Case Diagram



Câu 3: Vẽ sơ đồ phân rã Hệ thống ecomSys thành các dịch vụ và tương tác giữa chúng



Câu 4:Trình bày các dạng communication giữa các service với nhau (syschronous và asynchronous) với code và ví dụ

4.1. Giao tiếp đồng bộ (Synchronous Communication)

Trong giao tiếp đồng bộ, một dịch vụ gửi yêu cầu và đợi cho đến khi dịch vụ khác hoàn thành việc xử lý và trả về kết quả trước khi tiếp tục thực hiện các công việc khác. Quá trình này làm cho dịch vụ gửi
yêu cầu bị chặn (blocked) cho đến khi yêu cầu được xử lý xong, có thể tạo ra sự trễ trong hệ thống nếu yêu cầu đó mất nhiều thời gian để xử lý.

Ví dụ: Mô hình Request-Response trong HTTP, trong đó một client
gửi yêu cầu HTTP đến server và đợi cho đến khi server trả về phản hồi.

```python
def shipment_details_update(uname):
    ship_dict = {}
### It is used for getting data from payment info.
    user = paystat.objects.filter(username = uname)
    for data in user.values():
        data
    print(data)
    ship_dict['Product Id'] = data['product_id']
    ship_dict['Quantity'] = data['quantity']
    ship_dict['Payment Status'] = data['status']
    ship_dict['Transaction Id'] = data['id']
    ship_dict['Mobile Number'] = data['mobile']
    ### It is used for getting the user info.
    url = 'http://localhost:8000/api/userinfo'
    d1 = {}
    d1["User Name"] = data['username']
    print(d1)
    data = json.dumps(d1)
    print(data)
    headers = {'Content-Type': 'application/json'}
    response = requests.post(url, data=data, headers=headers)
    val1 = response.json()
    print(val1)
    ship_dict['First Name'] = val1['data']['First Name']
    ship_dict['Last Name'] = val1['data']['Last Name']
    ship_dict['Address'] = val1['data']['Address']
    ship_dict['Email Id'] = val1['data']['Email Id']
    ### Data is ready for calling the shipment_updates API.
    url = 'http://127.0.0.1:5000/api/shipment/shipment_updates'
    data = json.dumps(ship_dict)
    headers = {'Content-Type': 'application/json'}
    response = requests.post(url, data=data, headers=headers)
    api_resp = json.loads(response.content.decode('utf-8'))
    return api_resp
```

4.2. Giao tiếp không đồng bộ (Asynchronous Communication)

Trong giao tiếp không đồng bộ, dịch vụ gửi yêu cầu và tiếp tục thực hiện các công việc khác mà không cần phải chờ đợi kết quả từ dịch vụ đó. Kết quả có thể được trả về sau một khoảng thời gian, và dịch vụ gửi yêu cầu có thể tiếp tục xử lý kết quả mà không bị chặn.

Câu 5: Trình bày sử dụng các dạng communication giữa các service với code cho hệ ecomSys

5.1 CartService với MobileService, BookService, ClothesService: CartService gửi yêu cầu lấy sản phẩm tới MobileService, BookService, ClothesService: CartService để lưu vào rỏ hàng

5.2 CartService với UserService: CartService gửi yêu cầu tới UserService để lấy thông tin khách hàng

5.3 PaymentService với UserService: PaymentService gửi yêu cầu tới UserSerivce để lấy thông tin và trạng thái của khách hàng

5.4 PaymentService với ShipmentSerivce: PaymentService gửi yêu cầu tới ShipmentService để update trạng thái shipment